School Management System

DATA BASE

Submitted by:

Muhammad Yasir

Submitted to:

Syed Shayan Ali Shah

Semester: 4th BSDS(Group-B)

Session 2023 –2027

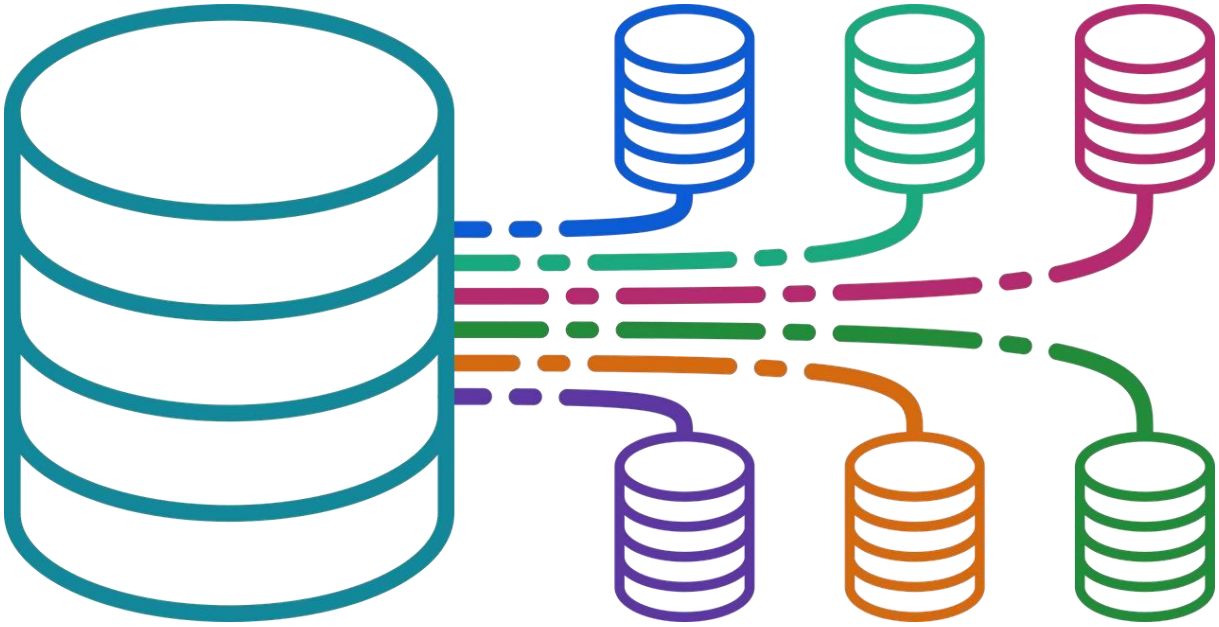
INSTITUTE OF MANAGEMENT SCIENCES

HAYATABAD, PESHAWAR (2025)

School Management System

DATABASE SYSTEMS

PROJECT REPORT



1. Introduction to the working of the system:

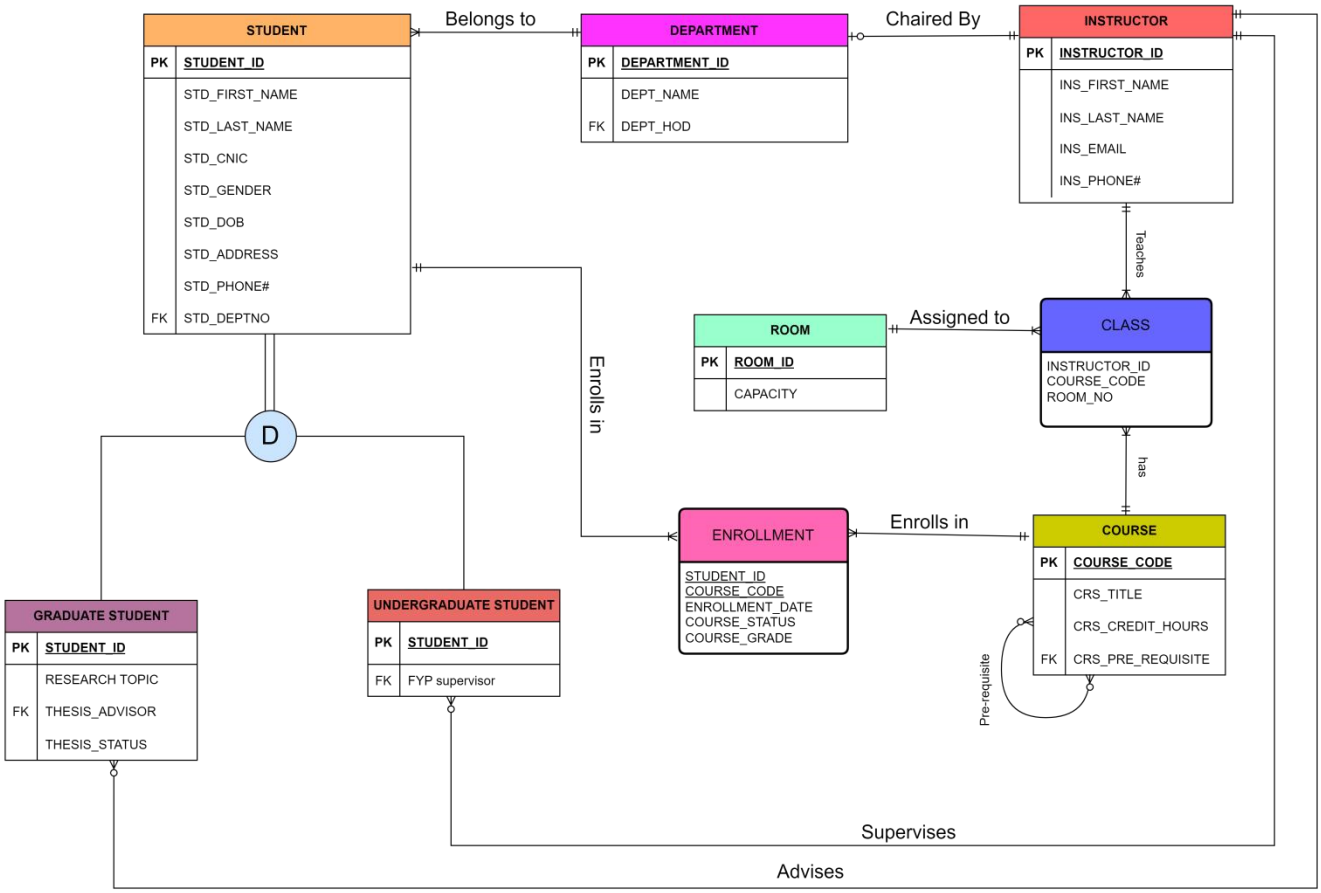
A database model has been developed at **IMSciences** to efficiently manage student records, including information about students, departments, instructors, and courses. This system streamlines operations, supports queries, and ensures future scalability.

2. Problems in the existing system:

A database model has been developed at **IMSciences** to efficiently manage academic records, encompassing information about **students, departments, instructors, and courses**. The system is designed to streamline administrative operations, facilitate complex queries, maintain data consistency, and support future scalability for institutional growth.

3. Entity - Relation Diagram:

c



4. ERD Transformation

Student

A strong entity

Student (Student_ID, First_Name, Last_Name, CNIC, Gender, DOB, Address, Phone#, DeptNo)

Graduate Student

Student's Subtype

Graduate Student (Student_ID, Thesis_Advisor_ID, Thesis_Status)

Undergraduate Student

Student's Subtype

Undergraduate Student (Student_ID, FYP_Advisor_ID)

Department

Strong Entity

Department (DeptNo , Dept_Name , Dept_HOD)

Instructor

A regular entity

Instructor (Instructor_ID, First_Name , Last_Name, Email, Phone#)

Room

A regular entity

Room (RoomNo , Capacity)

Course

Recursive relation

Course (Course_Code, Course_Title, Credit_Hours, Pre_Requisite_ID)

Class

An associative entity

Class (Instructor_ID, Course_Code, RoomNo)

Enrollment

An associative entity

Enrollment (Student_ID, Course_Code, Enrollment_Date, Course_Grade, Status)

NOTE:

_____ Indicates primary key

..... Indicates foreign key

5. CONSTRUCTION OF RELATIONAL SCHEMA

- ✧ TOP DOWN APPROACH
- ✧ BOTTOM UP APPROACH

TOPDOWNAPPROACH

Identified entities:

- ◆ Student
  - ■ Graduate students
  - ■ Undergraduate students
- ◆ Instructor
- ◆ Department
- ◆ Course
- ◆ Room
- ◆ Class ◆ Enrollment

Relations:

Student (Student_ID, First_Name, Last_Name, CNIC, Gender, DOB, Address, Phone#, DeptNo)

Graduate Student (Student_ID, Thesis_Advisor_ID, Thesis_Status)

Undergraduate Student (Student_ID, FYP_Advisor_ID)

Department (DeptNo , Dept_Name , Dept_HOD)

Instructor (Instructor_ID, First_Name , Last_Name, Email, Phone#)

Room (RoomNo , Capacity)

Course (Course_Code, Course_Title, Credit_Hours, Pre_Requisite_ID)

Class (Instructor_ID, Course_Code, RoomNo)

Enrollment (Student_ID, Course_Code, Enrollment_Date, Course_Grade, Status)

Normalization

Student (Student_ID, First_Name, Last_Name, CNIC, Gender, DOB, Address, Phone#, DeptNo)
1NF : Already in 1NF as there is no repeating group.
2NF : Already in 2NF as there is no Partial Functional Dependency.
3NF : Already in 3NF as there is no Transitive Dependency.

Graduate Student (Student_ID, Thesis_Advisor_ID, Thesis_Status)
1NF : Already in 1NF as there is no repeating group.
2NF : Already in 2NF as there is no Partial Functional Dependency.
3NF : Already in 3NF as there is no Transitive Dependency.

Undergraduate Student (Student_ID, FYP_Advisor_ID)
1NF : Already in 1NF as there is no repeating group.
2NF : Already in 2NF as there is no Partial Functional Dependency.
3NF : Already in 3NF as there is no Transitive Dependency.

Department (DeptNo , Dept_Name , Dept_HOD)
1NF : Already in 1NF as there is no repeating group.
2NF : Already in 2NF as there is no Partial Functional Dependency.
3NF : Already in 3NF as there is no Transitive Dependency.

Instructor (Instructor_ID, First_Name , Last_Name, Email, Phone#)
1NF : Already in 1NF as there is no repeating group.
2NF : Already in 2NF as there is no Partial Functional Dependency.
3NF : Already in 3NF as there is no Transitive Dependency.

Room (RoomNo , Capacity)
1NF : Already in 1NF as there is no repeating group.
2NF : Already in 2NF as there is no Partial Functional Dependency.
3NF : Already in 3NF as there is no Transitive Dependency.

Course (Course_Code, Course_Title, Credit_Hours, Pre_Requisite_ID)
1NF : A single course can have Multiple pre-requisites. So, there is a repeating group.
Shift the pre-requisite of course to another relation.
After removing repeating group:
Course (Course_Code, Course_Title, Credit_Hours)
Pre_Requisite_Course (Course_Code, Pre_Requisite_ID)
2NF : Already in 2NF as there is no Partial Functional Dependency.
3NF : Already in 3NF as there is no Transitive Dependency.

Class (Instructor_ID, Course_Code, RoomNo)

Normalization:

1NF : Already in 1NF as there is no repeating group.

2NF : Already in 2NF as there is no Partial Functional Dependency.

3NF : Already in 3NF as there is no Transitive Dependency.

Enrollment (Student_ID, Course_Code, Enrollment_Date, Course_Grade, Status)

1NF : Already in 1NF as there is no repeating group.

2NF : Already in 2NF as there is no Partial Functional Dependency.

3NF : Already in 3NF as there is no Transitive Dependency.

Result:

Now all relations are in 3NF, there's no further normalization needed. This indicates that the relations are structured well, with minimal redundancy and efficient data organization.

BOTTOMUPAPPROACH

Bulky relation comprising all attributes:

Relation → {Student_ID, Std_First_Name, Std_Last_Name, Std_CNIC, Std_Gender, Std_DOB, Std_Address, Std_Phone#, Std_DeptNo, DeptNo, Dept_Name ,Dept_HOD, Thesis_Advisor_ID, Thesis_Status, FYP_Advisor_ID, Instructor_ID, Ins_First_Name , Ins_Last_Name, Ins_Email, Ins_Phone#, RoomNo , Capacity, Course_Code, Course_Title, Credit_Hours, Pre_Requisite_ID,

Enrollment_ID, Enrollment_Date, Course_Grade, Status}

The relation consists of all the attributes in our present ERD. Now, we will construct a sub-relation from above and perform normalization.

Relation

Student(Student_ID,STD_First_name,Last_name,Std_CNIC,Gender,DOB,Address,Phone#,Deptno)

1NF : No repeating Group

2NF : No Partial Functional Dependency

3NF : No Transitive Dependency

Subtypes:

Graduate Student (Student_ID, Thesis_Advisor_ID, Thesis_Status)

*1*NF *:* There is no repeating group

*2*NF *:* Already in 2NF as there is no Partial Functional Dependency.
*3*NF : Already in 3NF as there is no Transitive Dependency.

Undergraduate Student (Student_ID, FYP_Advisor_ID)

*1*NF *:* Already in 1NF as there is no repeating group.
*2*NF *:* Already in 2NF as there is no Partial Functional Dependency.
*3*NF : Already in 3NF as there is no Transitive Dependency.

DEPARTMENT (Deptno,Dept_Name,Dept_HOD)

*1*NF *:* Already in 1NF as there are no multivalued attributes.
*2*NF *:* Already in 2NF as there is no Partial Functional Dependency.
*3*NF : Already in 3NF as there is no Transitive Dependency.

Instructor (Instructor_ID, First_Name , Last_Name, Email, Phone#)

*1*NF *:* Already in 1NF as no duplicating values.
*2*NF *:* Already in 2NF as atomic primary key.
*3*NF : Already in 3NF as no non-key attributes determine other attributes.

Room (RoomNo , Capacity)

*1*NF *:* No duplicate data as there is only one Primary key *2*NF *:*
Already in 2NF as there is no Partial Functional Dependency.
*3*NF : Already in 3NF as there is no Transitive Dependency.

Course (Course_Code, Course_Title, Credit_Hours, Pre_Requisite_ID)

There may be multiple pre-requisite for one course.
So,It does not hold 1NF requirements

After 1NF

Course(Course_Code, Course_Title, Credit_Hours)
Course_Pre_Req (Course_Code, Pre_Requisite_ID)

*2*NF *:* Already in 2NF as there is no Partial Functional Dependency.
*3*NF : Already in 3NF as there is no Transitive Dependency.

Class (Instructor_ID, Course_Code, RoomNo)

*1*NF *:* Already in 1NF as there is no repeating group.
*2*NF *:* Already in 2NF as there is no Partial Functional Dependency.
*3*NF : Already in 3NF as there is no Transitive Dependency.

Enrollment (Student_ID, Course_Code, Enrollment_Date, Course_Grade, Status) *1*NF
: Already in 1NF as there is no repeating group.
*2*NF *:* Already in 2NF as there is no Partial Functional Dependency.
*3*NF : Already in 3NF as there is no Transitive Dependency.

Connectivity Table:

Entity	Relationship	Connectivity	Entity
Student	Is a	1:1	Graduate Student
Student	Is a	1:1	Under Graduate Student
Department	Has	1:M	Student
Instructor	Has	1:1	department
Instructor	Has	1:M	Graduate Student
Instructor	Has	1:M	Under graduate student
Student	Has	1:M	Course
Course	Has	1:M	Course
Course	Has	1:M	Student
Room	Has	1:M	Class
Instructor	Has	1:M	Class

6. Description of relations

Student

Attribute	Type	Size	Constraints
STUDENT_ID	CHAR	10	PRIMARY KEY
FIRST_NAME	VARCHAR2	20	NOT NULL
LAST_NAME	VARCHAR2	20	NOT NULL
CNIC	CHAR	13	UNIQUE
GENDER	CHAR	1	M OR F
DOB	DATE		
ADDRESS	VARCHAR2	50	
PHONE#	CHAR	13	NOT NULL , UNIQUE
DEPTNO	NUMBER		REFERENCE TO DEPARTMENT

Graduate Student

Attribute	Type	Size	Constraints
STUDENT_ID	CHAR	10	PRIMARY KEY, REFERENCE TO STUDENT
THESIS_ADVISOR	CHAR	10	REFERENCE TO INSTRUCTOR
THESIS_STATUS	VARCHAR2	20	'COMPLETE' OR 'IN PROGRESS'

Under Graduate Student

Attribute	Type	Size	Constraints
STUDENT_ID	CHAR	10	PRIMARY KEY, REFERENCE TO STUDENT
FYP_ADVISOR	CHAR	10	REFERENCE TO INSTRUCTOR

Department

Attribute	Type	Size	Constraints
DEPTNO	NUMBER		PRIMARY KEY
DEPTNAME	VARCHAR2	50	NOT NULL
HOD	CHAR	10	REFERENCE TO INSTRUCTOR

Instructor

Attribute	Type	Size	Constraints
INSTRUCTOR_ID	CHAR	10	PRIMARY KEY
FIRST_NAME	VARCHAR2	20	NOT NULL
LAST_NAME	VARCHAR2	20	NOT NULL
EMAIL	VARCHAR2	30	UNIQUE
PHONE#	CHAR	13	NOT NULL , UNIQUE

Room

Attribute	Type	Size	Constraints
ROOMNO	NUMBER		PRIMARY KEY
CAPACITY	NUMBER		POSITIVE INTEGER

Course

Attribute	Type	Size	Constraints
COURSE_CODE	VARCHAR2	10	PRIMARY KEY
COURSE_TITLE	VARCHAR2	50	NOT NULL
CREDIT_HOURS	NUMBER		BETWEEN 0.5 AND 3

Pre_Requisite_Course

Attribute	Type	Size	Constraints
COURSE_CODE	VARCHAR2	10	PRIMARY KEY, REFERENCE TO COURSE
PRE_REQ	VARCHAR2	10	PRIMARY KEY, REFERENCE TO COURSE

Class

Attribute	Type	Size	Constraints
INSTRUCTOR_ID	CHAR	10	PRIMARY KEY, REFERENCE TO INSTRUCTOR
COURSE_CODE	VARCHAR2	10	PRIMARY KEY, REFERENCE TO COURSE
ROOM_NO	NUMBER		REFERENCE TO ROOM

Enrollment

Attribute	Type	Size	Constraints
STUDENT_ID	CHAR	10	PRIMARY KEY, REFERENCE TO STUDENT
COURSE_CODE	VARCHAR2	10	PRIMARY KEY,REFERENCE TO COURSE
ENROLLMENT_DATE	DATE		
GRADE	CHAR	1	‘A’,’B’,’C’,’D’ OR F
STATUS	CHAR	4	‘PASS’ OR ‘FAIL’

7. CREATE TABLE statements for all relations

Student

```
CREATE TABLE Student
(
    STUDENT_ID CHAR(10) CONSTRAINT pk_student PRIMARY KEY,
    FIRST_NAME VARCHAR(20) CONSTRAINT nn_first_name NOT NULL,
    LAST_NAME VARCHAR(20) CONSTRAINT nn_last_name NOT NULL,
    CNIC CHAR(13) CONSTRAINT uq_cnic UNIQUE,
    GENDER CHAR(1) CONSTRAINT ck_gender CHECK (GENDER IN ('M', 'F')),
    DOB DATE,
    ADDRESS VARCHAR(50),
    PHONE CHAR(13) CONSTRAINT nn_phone NOT NULL,
```

```
    DEPTNO NUMBER CONSTRAINT fk_deptno REFERENCES Department(DEPTNO));
```

Data Output Messages Notifications

	student_id [PK] character (10)	first_name character varying (20)	last_name character varying (20)	cnic character (13)	gender character (1)	dob date	address character varying (50)	phone character (13)	deptno integer
1	S001	Eve	Davis	3520123456789	F	2000-04-15	123 Main St	555-111-2222	1
2	S002	Frank	Miller	3529876543210	M	1999-11-30	456 Oak Ave	555-333-4444	1
3	S003	Grace	Wilson	3525647382910	F	2001-07-20	789 Pine Rd	555-555-6666	2
4	S004	Henry	Moore	3520123498765	M	2000-12-10	101 Maple St	555-777-8888	3
5	S0031	Ali	Khan	1234512345671	M	2000-01-01	Peshawar	03001234567	1
6	S0032	Sara	Yousaf	1234512345672	F	1999-02-02	Lahore	03001234568	1
7	S0033	Hamza	Ahmed	1234512345673	M	2001-03-03	Karachi	03001234569	1

Graduate Student

```
CREATE TABLE Graduate_Student
(
    STUDENT_ID CHAR(10) CONSTRAINT pk_graduate_student PRIMARY KEY REFERENCES Student(STUDENT_ID),
    THESIS_ADVISOR CHAR(10) CONSTRAINT fk_thesis_advisor REFERENCES Instructor(INSTRUCTOR_ID),
    THESIS_STATUS VARCHAR(20) CONSTRAINT ck_thesis_status CHECK (THESIS_STATUS IN ('COMPLETE', 'IN PROGRESS'))
);
```

Data Output Messages Notifications

	student_id [PK] character (10)	thesis_advisor character (10)	thesis_status character varying (20)
1	S002	I001	IN PROGRESS
2	S004	I003	COMPLETE

Under Graduate Student

```
CREATE TABLE Undergraduate_Student
(
    STUDENT_ID CHAR(10) CONSTRAINT pk_undergraduate_student PRIMARY KEY REFERENCES Student(STUDENT_ID),
    FYP_ADVISOR CHAR(10) CONSTRAINT fk_fyp_advisor REFERENCES Instructor(INSTRUCTOR_ID)
);
```

Data Output Messages Notifications

	student_id [PK] character (10)	fyp_advisor character (10)
1	S001	I001
2	S003	I002

Department

```
CREATE TABLE department (
    deptno INTEGER CONSTRAINT pk_department PRIMARY KEY,
    deptname VARCHAR(50) NOT NULL CONSTRAINT nn_deptname,
    hod CHAR(10) CONSTRAINT fk_hod REFERENCES instructor(instructor_id)
);
```

Data Output Messages Notifications

	deptno [PK] integer	deptname character varying (50)	hod character (10)
1	1	Computer Science	I001
2	2	Electrical Engineering	I002
3	3	Business Administration	I003

Instructor

```
CREATE TABLE instructor (
    instructor_id CHAR(10) CONSTRAINT pk_instructor PRIMARY KEY,
    first_name VARCHAR(20) NOT NULL CONSTRAINT nn_instructor_first_name,
    last_name VARCHAR(20) NOT NULL CONSTRAINT nn_instructor_last_name,
    email VARCHAR(30) CONSTRAINT uq_instructor_email UNIQUE,
```

```
phone CHAR(13) NOT NULL CONSTRAINT nn_instructor_phone);
```

Data Output Messages Notifications

	instructor_id [PK] character (10)	first_name character varying (20)	last_name character varying (20)	email character varying (30)	phone character (13)
1	I001	Alice	Smith	alice.smith@university.edu	123-456-7890
2	I002	Bob	Johnson	bob.johnson@university.edu	234-567-8901
3	I003	Carol	Williams	carol.williams@university.edu	345-678-9012
4	I004	David	Brown	david.brown@university.edu	456-789-0123

Room

```
CREATE TABLE room (
roomno INTEGER CONSTRAINT pk_room PRIMARY KEY,

capacity INTEGER CONSTRAINT ck_capacity CHECK (capacity > 0)
);
```

Data Output Messages Notifications

	roomno [PK] integer	capacity integer
1	101	40
2	102	30
3	201	50
4	202	25

Course

```
CREATE TABLE Course
(
    COURSE_CODE VARCHAR2(10) CONSTRAINT pk_course PRIMARY KEY,
    COURSE_TITLE VARCHAR2(50) CONSTRAINT nn_course_title NOT NULL,
    CREDIT_HOURS NUMBER CONSTRAINT ck_credit_hours CHECK (CREDIT_HOURS
    BETWEEN 0.5 AND 3) );
```

Data Output Messages Notifications

	course_code [PK] character varying (10)	course_title character varying (50)	credit_hours numeric (3,1)
1	CS101	Introduction to Computer Science	3.0
2	EE101	Basic Electrical Engineering	3.0
3	BA101	Principles of Management	3.0
4	CS201	Data Structures	3.0
5	CSE101	Intro to Programming	3.0
6	CSE102	Data Structures	3.0
7	CSE103	Databases	3.0

Pre_Requisite_Course

```
CREATE TABLE pre_requisite_course (
    course_code VARCHAR(10) CONSTRAINT fk_course_code REFERENCES
course(course_code),
    pre_req VARCHAR(10) CONSTRAINT fk_pre_req REFERENCES
course(course_code),
    CONSTRAINT pk_pre_requisite_course PRIMARY KEY (course_code, pre_req));
```

Data Output Messages Notifications

	course_code [PK] character varying (10)	pre_req [PK] character varying (10)
1	CS201	CS101
2	EE101	BA101
3	CSE102	CSE101

Class

```
CREATE TABLE class (
    instructor_id CHAR(10) CONSTRAINT fk_class_instructor REFERENCES
instructor(instructor_id),
course_code VARCHAR(10) CONSTRAINT fk_class_course REFERENCES
course(course_code),
    room_no INTEGER CONSTRAINT fk_class_room REFERENCES room(roomno)
    CONSTRAINT pk_class PRIMARY KEY (instructor_id, course_code));
```

Data Output Messages Notifications

	instructor_id [PK] character (10)	course_code [PK] character varying (10)	room_no integer
1	I001	CS101	101
2	I002	EE101	102
3	I003	BA101	201
4	I001	CS201	202

Enrollment

```
CREATE TABLE enrollment (
    student_id CHAR(10) CONSTRAINT fk_enrollment_student REFERENCES
student(student_id),
    course_code VARCHAR(10) CONSTRAINT fk_enrollment_course REFERENCES
course(course_code),
    enrollment_date DATE,
    grade CHAR(1) CONSTRAINT ck_grade CHECK (grade IN ('A', 'B', 'C', 'D',
'F')),
    status CHAR(4) CONSTRAINT ck_status CHECK (status IN ('PASS', 'FAIL')),
    CONSTRAINT pk_enrollment PRIMARY KEY (student_id, course_code)
);
```

Data Output Messages Notifications

	student_id [PK] character (10)	course_code [PK] character varying (10)	enrollment_date date	grade character (1)	status character (4)

8. Views of relational schema

1. Student Details:

```
CREATE OR REPLACE VIEW student_details AS
SELECT
    S.STUDENT_ID,
    S.FIRST_NAME || ' ' || S.LAST_NAME AS FULL_NAME,
    S.CNIC,
    S.GENDER,
    S.DOB,
    S.ADDRESS,
    S.PHONE,
    D.DEPTNAME AS DEPARTMENT_NAME,
    H.FIRST_NAME || ' ' || H.LAST_NAME AS DEPARTMENT_HEAD
FROM
    Student S
JOIN Department D ON S.DEPTNO = D.DEPTNO

JOIN Instructor H ON D.HOD =

H.INSTRUCTOR_ID; select * from

student_details
```

2. Enrollment Details

```
CREATE VIEW Enrollment_Details AS
SELECT
    e.STUDENT_ID,
    s.FIRST_NAME,
    s.LAST_NAME,
    e.COURSE_CODE,
    c.COURSE_TITLE,
    e.ENROLLMENT_DATE,
    e.GRADE,
    e.STATUS
FROM Enrollment e
JOIN Student s ON e.STUDENT_ID = s.STUDENT_ID

JOIN Course c ON e.COURSE_CODE = c.COURSE_CODE

select * from enrollment_details
```

3. Graduate Students Details:

```
CREATE OR REPLACE VIEW Graduate_Student_Details AS
SELECT
    gs.STUDENT_ID,
    CONCAT(s.FIRST_NAME,CONCAT(' ',s.LAST_NAME)) "STUDENT NAME",
    s.CNIC,
    s.GENDER,
    s.DOB,
    s.ADDRESS,
    s.PHONE,
    CONCAT(i.FIRST_NAME,CONCAT(' ',i.LAST_NAME)) "ÄDVISOR NAME",
    gs.THESIS_STATUS
FROM Graduate_Student gs
JOIN Student s ON gs.STUDENT_ID = s.STUDENT_ID
JOIN Instructor i ON gs.THESIS_ADVISOR = i.INSTRUCTOR_ID;
```

4. Pre-Requisite Courses Details:

```
CREATE OR REPLACE VIEW Pre_Requisite_Course_Details AS
SELECT
    c.COURSE_TITLE AS COURSE_TITLE,
    prc2.COURSE_TITLE AS PRE_REQ_TITLE
FROM Pre_Requisite_Course prc
JOIN Course c ON prc.COURSE_CODE = c.COURSE_CODE
JOIN Course prc2 ON prc.PRE_REQ = prc2.COURSE_CODE;
```

9. Relational Data Model showing associations

10. Five Common Reports

1. QUERY TO RETRIEVE STUDENTS WHOSE THESIS IS COMPLETE

```
SELECT
  S.FIRST_NAME || ' ' || S.LAST_NAME AS NAME,
  I.FIRST_NAME || ' ' || I.LAST_NAME AS ADVISOR,
  G.THESIS_STATUS
FROM STUDENT S
JOIN GRADUATE_STUDENT G ON G.STUDENT_ID = S.STUDENT_ID
JOIN INSTRUCTOR I ON I.INSTRUCTOR_ID = G.THESIS_ADVISOR
WHERE G.THESIS_STATUS = 'COMPLETE';
```

2. QUERY TO RETRIEVE ROOMS THAT ARE FREE YET

```
SELECT R.ROOMNO, R.CAPACITY
FROM ROOM R
LEFT JOIN CLASS C ON R.ROOMNO = C.ROOM_NO
WHERE C.ROOM_NO IS NULL;
```

3. QUERY TO RETRIEVE DEPARTMENT ID ,NAME ,HEAD AND NUMBER OF STUDENTS IN THAT DEPARTMENT

```
SELECT
    d.DEPTNO,
    d.DEPTNAME,
    i.FIRST_NAME || ' ' || i.LAST_NAME AS HEAD_NAME,
    COUNT(s.STUDENT_ID) AS STUDENT_COUNT
FROM Department d
JOIN Instructor i ON d.HOD = i.INSTRUCTOR_ID
LEFT JOIN Student s ON d.DEPTNO = s.DEPTNO
GROUP BY d.DEPTNO, d.DEPTNAME, i.FIRST_NAME, i.LAST_NAME;
```

4. LIST INSTRUCTORS WHO ARE SUPERVISING AN FYP

```
SELECT DISTINCT
  I.INSTRUCTOR_ID,
  I.FIRST_NAME || ' ' || I.LAST_NAME AS NAME
FROM Instructor I
JOIN Undergraduate_Student U ON I.INSTRUCTOR_ID = U.FYP_ADVISOR;
```

5. STUDENT WHO ARE FAILED IN ANY SUBJECT

```
SELECT
  S.STUDENT_ID,
  S.FIRST_NAME || ' ' || S.LAST_NAME AS NAME,
  C.COURSE_TITLE,
  I.FIRST_NAME || ' ' || I.LAST_NAME AS TEACHER_NAME
FROM STUDENT S
JOIN ENROLLMENT E ON S.STUDENT_ID = E.STUDENT_ID
```

```
JOIN COURSE C ON C.COURSE_CODE = E.COURSE_CODE
JOIN CLASS CL ON E.COURSE_CODE = CL.COURSE_CODE
JOIN INSTRUCTOR I ON I.INSTRUCTOR_ID = CL.INSTRUCTOR_ID
WHERE E.GRADE = 'F';
```

11. Procedures

1. PROCEDURE TO SHOW THE RESULT OF SPECIFIC STUDENT

```
CREATE OR REPLACE PROCEDURE Student_Result(p_STUDENT_ID
CHAR)
LANGUAGE plpgsql
AS $$
DECLARE
    rec RECORD;
    no_enrollments BOOLEAN := TRUE;
BEGIN
    FOR rec IN
        SELECT e.COURSE_CODE, c.COURSE_TITLE, e.GRADE,
e.STATUS
        FROM Enrollment e
        JOIN Course c ON e.COURSE_CODE = c.COURSE_CODE
        WHERE e.STUDENT_ID = p_STUDENT_ID
    LOOP
        no_enrollments := FALSE;
        RAISE NOTICE 'Course Code: %', rec.COURSE_CODE;
        RAISE NOTICE 'Course Title: %',
rec.COURSE_TITLE;
        RAISE NOTICE 'Grade: %', rec.GRADE;
        RAISE NOTICE 'Status: %', rec.STATUS;
        RAISE NOTICE '-----------------------------';
    END LOOP;

    IF no_enrollments THEN
        RAISE NOTICE 'No enrollments found for the given
Student ID.';
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        RAISE NOTICE 'An error occurred: %', SQLERRM;
END;
$$;
```

2. Procedure to Insert a student

```
CREATE OR REPLACE PROCEDURE
    InsertStudent( p_STUDENT_ID CHAR,
    p_FIRST_NAME VARCHAR2, p_LAST_NAME
    VARCHAR2, p_CNIC CHAR, p_GENDER CHAR,
    p_DOB DATE, p_ADDRESS VARCHAR2,
    p_PHONE CHAR,
```

```
    p_DEPTNO NUMBER
) IS
BEGIN
    INSERT INTO Student (STUDENT_ID,FIRST_NAME,LAST_NAME,CNIC,
        GENDER,DOB,ADDRESS,PHONE,DEPTNO) VALUES
(p_STUDENT_ID, p_FIRST_NAME,p_LAST_NAME,p_CNIC,p_GENDER, p_DOB,
 p_ADDRESS,p_PHONE,p_DEPTNO);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No data found.');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Too many Rows');
END;
/
```

3. PROCEDURE TO INSERT A RECORD IN ENROLLMENT TABLE AND CALCULATING GRADE BY MARKS

```
CREATE
OR REPLACE PROCEDURE InsertGrade(
    p_student_id CHAR,
    p_course_id VARCHAR,
    p_marks NUMERIC
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_grade CHAR(1);
    v_status CHAR(4);
BEGIN
    IF p_marks > 85 THEN
        v_grade := 'A';
        v_status := 'PASS';
    ELSIF p_marks > 75 THEN
        v_grade := 'B';
        v_status := 'PASS';
    ELSIF p_marks > 65 THEN
        v_grade := 'C';
        v_status := 'PASS';
    ELSIF p_marks > 50 THEN
        v_grade := 'D';
        v_status := 'PASS';
    ELSE
        v_grade := 'F';
        v_status := 'FAIL';
```

```
    END IF;

    INSERT INTO Enrollment
(STUDENT_ID, COURSE_CODE,
ENROLLMENT_DATE, GRADE, STATUS)
    VALUES (p_student_id, p_course_id,
CURRENT_DATE, v_grade, v_status);

EXCEPTION
    WHEN OTHERS THEN
        RAISE NOTICE 'Error occurred:
%', SQLERRM;
END;
$$;
```

12. Functions

1. FUNCTION TO GET AVERAGE GRADE POINT IN A COURSE

```
CREATE OR REPLACE FUNCTION
GetAVGGradePoint(p_course_code VARCHAR)
RETURNS NUMERIC
LANGUAGE plpgsql
AS $$
DECLARE
    v_average_grade NUMERIC;
BEGIN
    SELECT AVG(
        CASE GRADE
            WHEN 'A' THEN 4.0
            WHEN 'B' THEN 3.0
            WHEN 'C' THEN 2.0
            WHEN 'D' THEN 1.0
            WHEN 'F' THEN 0.0
            ELSE NULL
        END
    )
    INTO v_average_grade
    FROM Enrollment
    WHERE COURSE_CODE = p_course_code;

    RETURN v_average_grade;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN NULL;
END;
$$;
```

2. FUNCTION TO GET NUMBER OF COURSES BEING TAUGT BY A SPECIFIC INSTRUCTOR

```
CREATE OR REPLACE FUNCTION GetClassCountForTeacher(p_instructor_id
CHAR)
RETURNS INTEGER
LANGUAGE plpgsql
AS $$
DECLARE
    v_class_count INTEGER;
BEGIN
    SELECT COUNT(*)
    INTO v_class_count
    FROM Class
    WHERE INSTRUCTOR_ID = p_instructor_id;

    RETURN v_class_count;
EXCEPTION
    WHEN OTHERS THEN
        RETURN -1;
END;
$$;
```

3. FUNCTION TO COUNT THE NUMBER OF GRADUATE STUDENTS IN A DEPARTMENT

```
CREATE OR REPLACE FUNCTION GetTotalGraduateStudentsInDept(p_deptno
INTEGER)
RETURNS INTEGER
LANGUAGE plpgsql
AS $$
DECLARE
    v_student_count INTEGER;
BEGIN
    SELECT COUNT(*)
    INTO v_student_count
    FROM Graduate_Student gs
    JOIN Student s ON gs.STUDENT_ID = s.STUDENT_ID
    WHERE s.DEPTNO = p_deptno;

    RETURN v_student_count;
EXCEPTION
    WHEN OTHERS THEN
        RETURN -1;
END;
$$;
```

13. Triggers

Trigger that no more than 20 students are enrolled in a course

```
CREATE OR REPLACE FUNCTION trg_CheckCourseEnrollmentLimit()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
DECLARE
    v_enrolled_students INTEGER;
BEGIN
    SELECT COUNT(*)
    INTO v_enrolled_students
    FROM Enrollment
    WHERE COURSE_CODE = NEW.COURSE_CODE;

    IF v_enrolled_students >= 20 THEN
        RAISE EXCEPTION 'Enrollment limit exceeded for this course.
No more than 20 students can be enrolled.';
    END IF;

    RETURN NEW;
END;
$$;

CREATE TRIGGER trg_CheckCourseEnrollmentLimit
BEFORE INSERT ON Enrollment
FOR EACH ROW
EXECUTE FUNCTION trg_CheckCourseEnrollmentLimit();;
/
```

Trigger to check if student has passed pre req

```
CREATE OR REPLACE FUNCTION trg_EnforcePrereqCompletion()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
DECLARE
    v_prereq_completed INTEGER;
BEGIN
    SELECT COUNT(*)
    INTO v_prereq_completed
    FROM Enrollment e
    JOIN Pre_Requisite_Course p ON e.COURSE_CODE = p.PRE_REQ
    WHERE e.STUDENT_ID = NEW.STUDENT_ID
      AND p.COURSE_CODE = NEW.COURSE_CODE
      AND e.STATUS = 'PASS';

    IF v_prereq_completed = 0 THEN
        RAISE EXCEPTION 'Prerequisite courses not completed.';
```

```
    END IF;

    RETURN NEW;
END;
$$;

CREATE TRIGGER trg_EnforcePrereqCompletion
BEFORE INSERT ON Enrollment
FOR EACH ROW
EXECUTE FUNCTION trg_EnforcePrereqCompletion();
```